Machine Learning

Classification using Neural Networks

Content

- Linear Classifiers
  - Perceptron
  - Training
  - Limitations
- Multilayer Perceptron
  - Cost function
  - Training with Backpropagation
- Generalization
  - Early stopping
  - Regularization

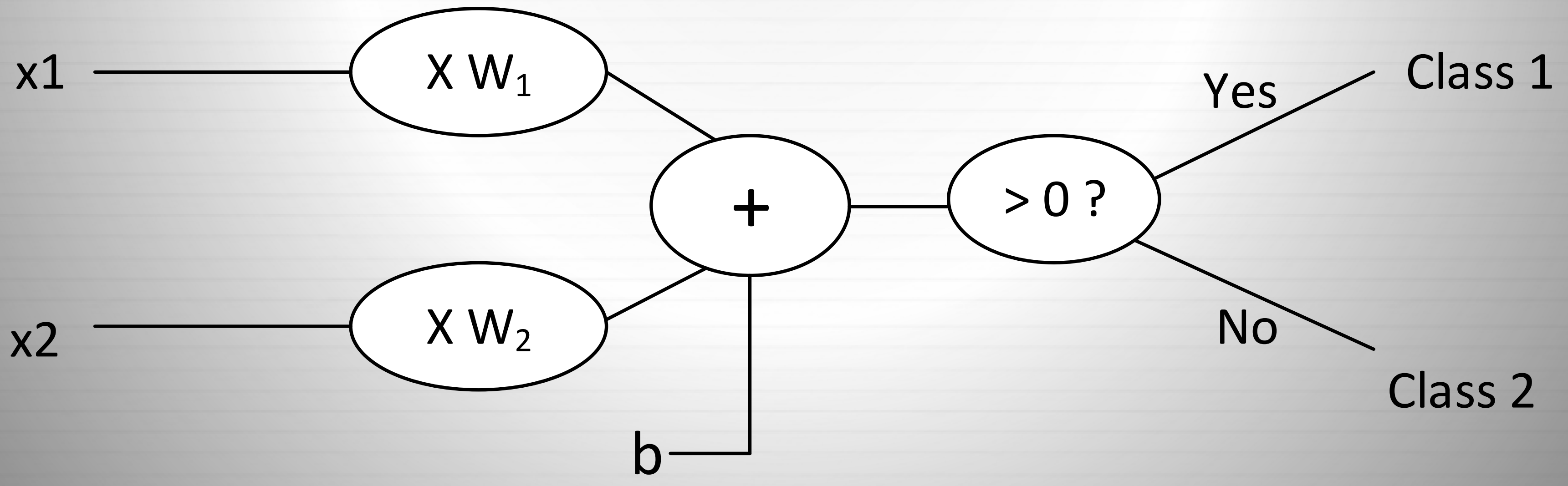
Linear Classifier

- A linear classifier divides the feature space into two parts
- A linear classifier is a linear combination of features as:

$$w_1x_1 + w_2x_2 + \ldots + w_nx_n + b = f(x)$$

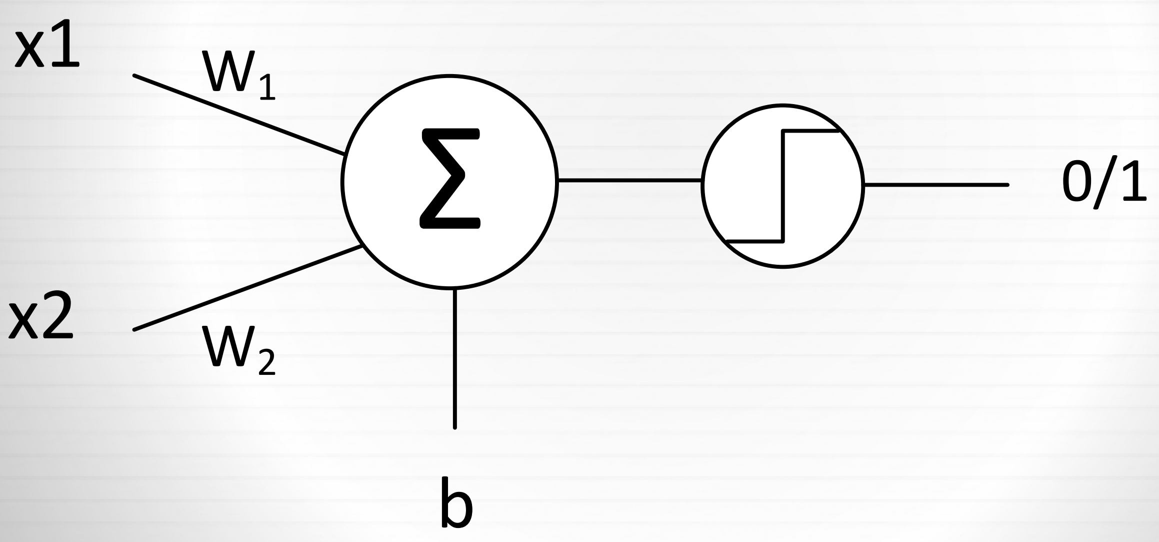
Linear Classifiers

- If f(X) > 0 the sample is from class 1
- Else, it is from class 2
- The linear classifier can be visualized as:



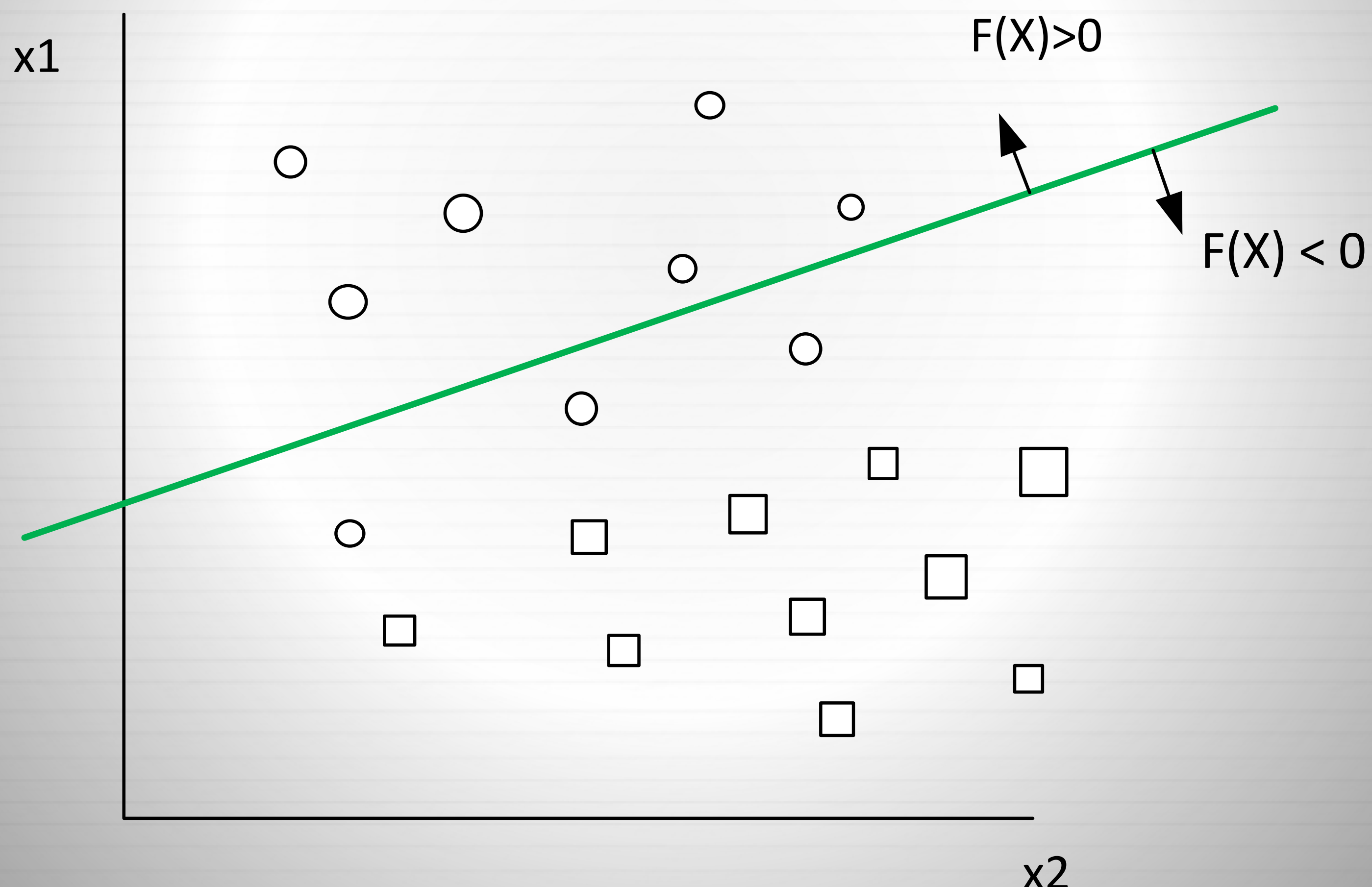
Linear Classifier Model

- The model can be simplified as:

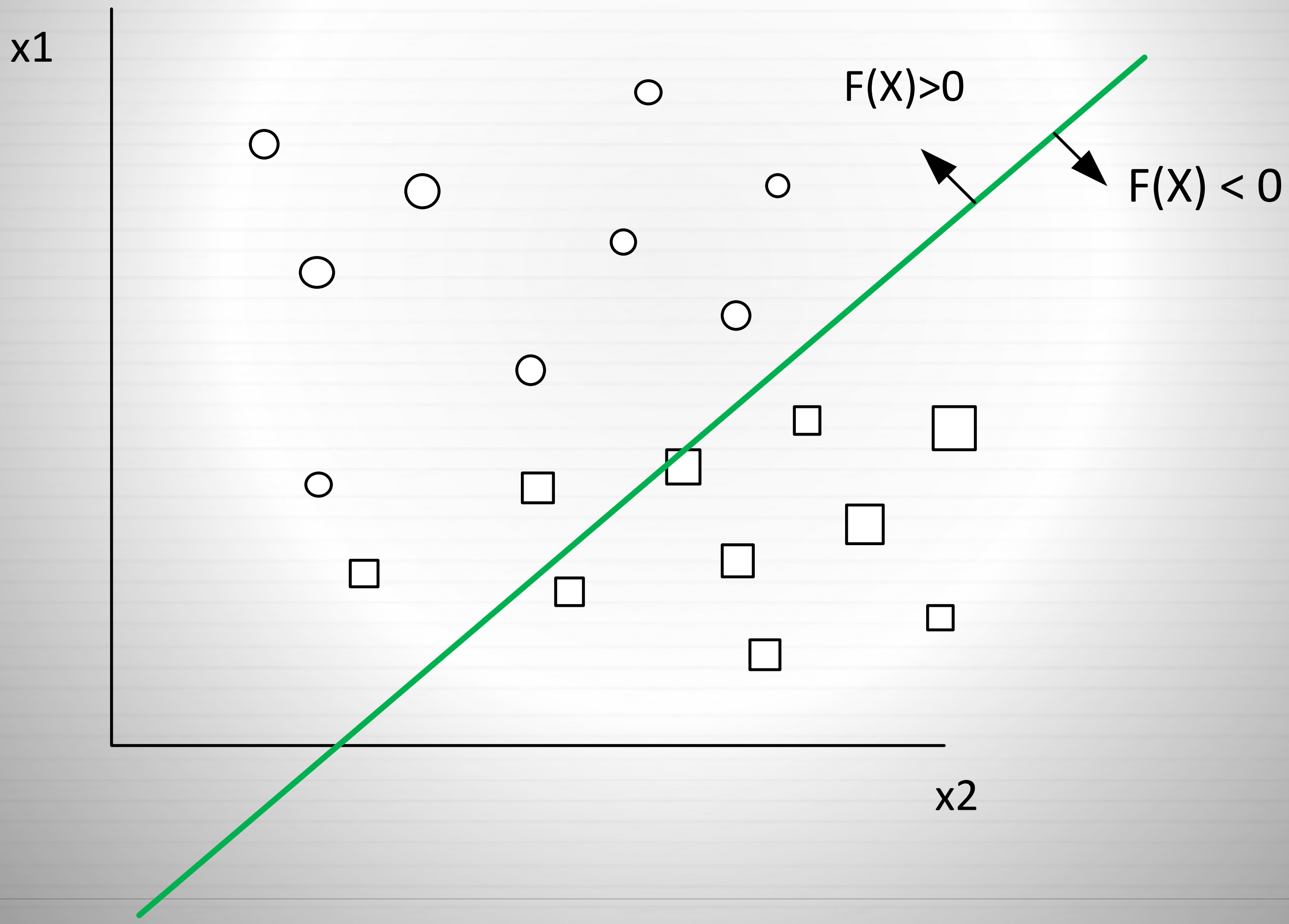


- This model is called **Perceptron**

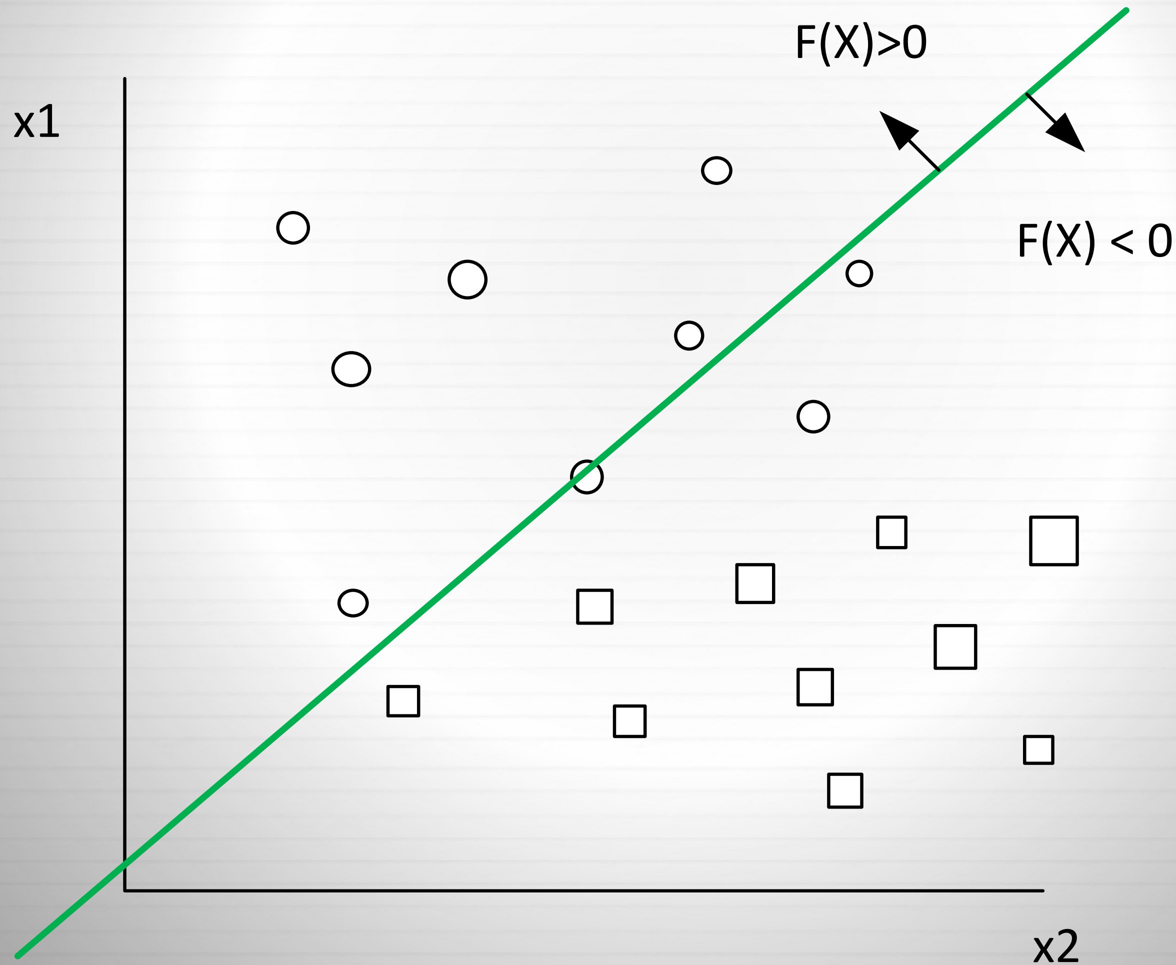
A Linear Classifier



A Linear Classifier



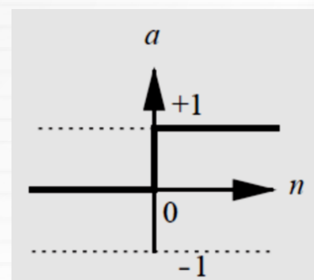
A Linear Classifier

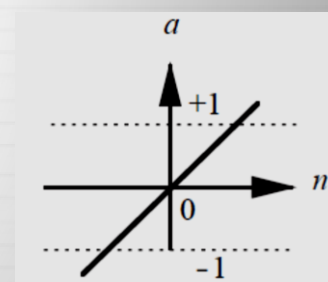


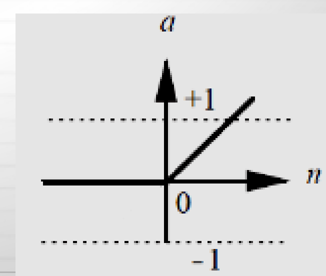
Transfer Function

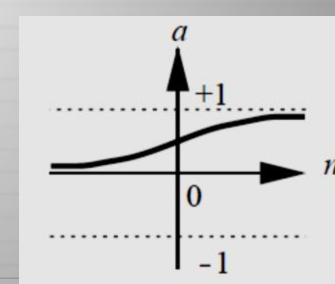
- The summed up input, often referred to as the *net input*, goes into a *transfer function* , which produces the neuron (perceptron) output

- Four of the most commonly used functions are:
  - *hard limit transfer function*
  - *linear transfer function*
  - *ReLu transfer function*
  - *log-sigmoid transfer function*









Perceptron

- Perceptron is a supervised linear classifier
- Perceptron in trained by changing its weight values
- To train Perceptron, we compare actual output (a) with true output (t)

Perceptron Learning

- Repeat over training data
  - If t=1 and a=0 then

    $$w_{new} = w_{old} + p, \quad b_{new} = b_{old} + 1$$
  - If t=0 and a=1 then

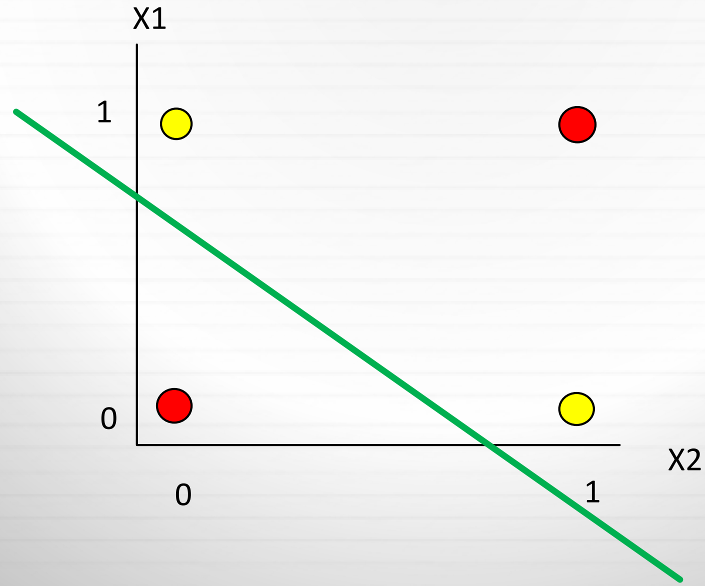
    $$w_{new} = w_{old} - p, \quad b_{new} = b_{old} - 1$$
  - If t=a then $w_{new} = w_{old}$

Perceptron Learning

- Learning continues until all data samples are tested.
- This is called one epoch.
- Training stops if
  - Weights do not change in an epoch, OR
  - We have reached to the upper limit of epochs

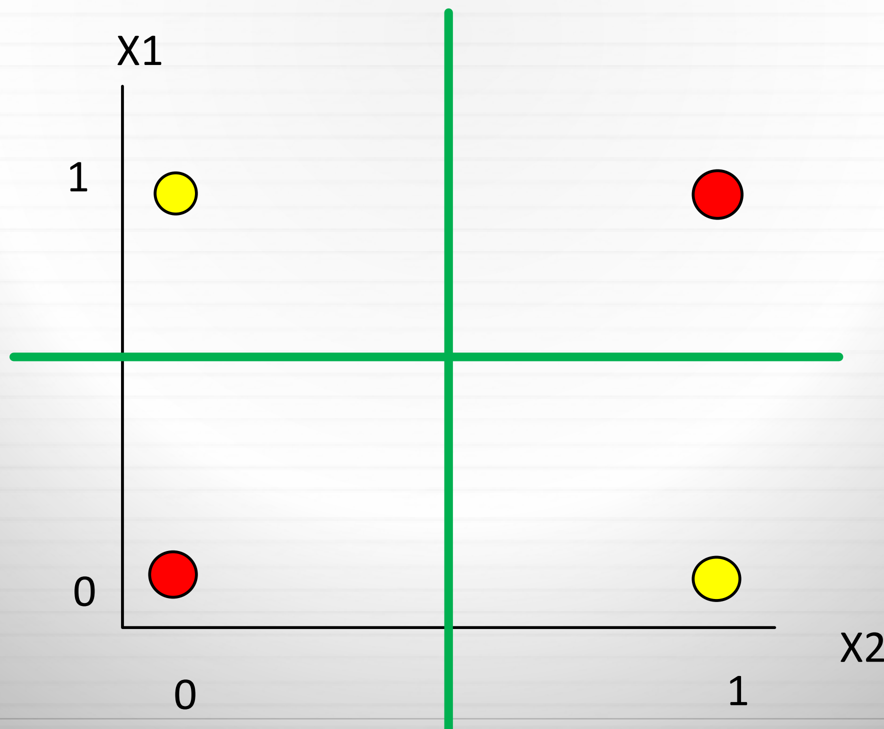
Limitations

- If the classes are not linearly separable, Perceptron will fail to classify them.
- Example of non-linearly separable data is XOR function

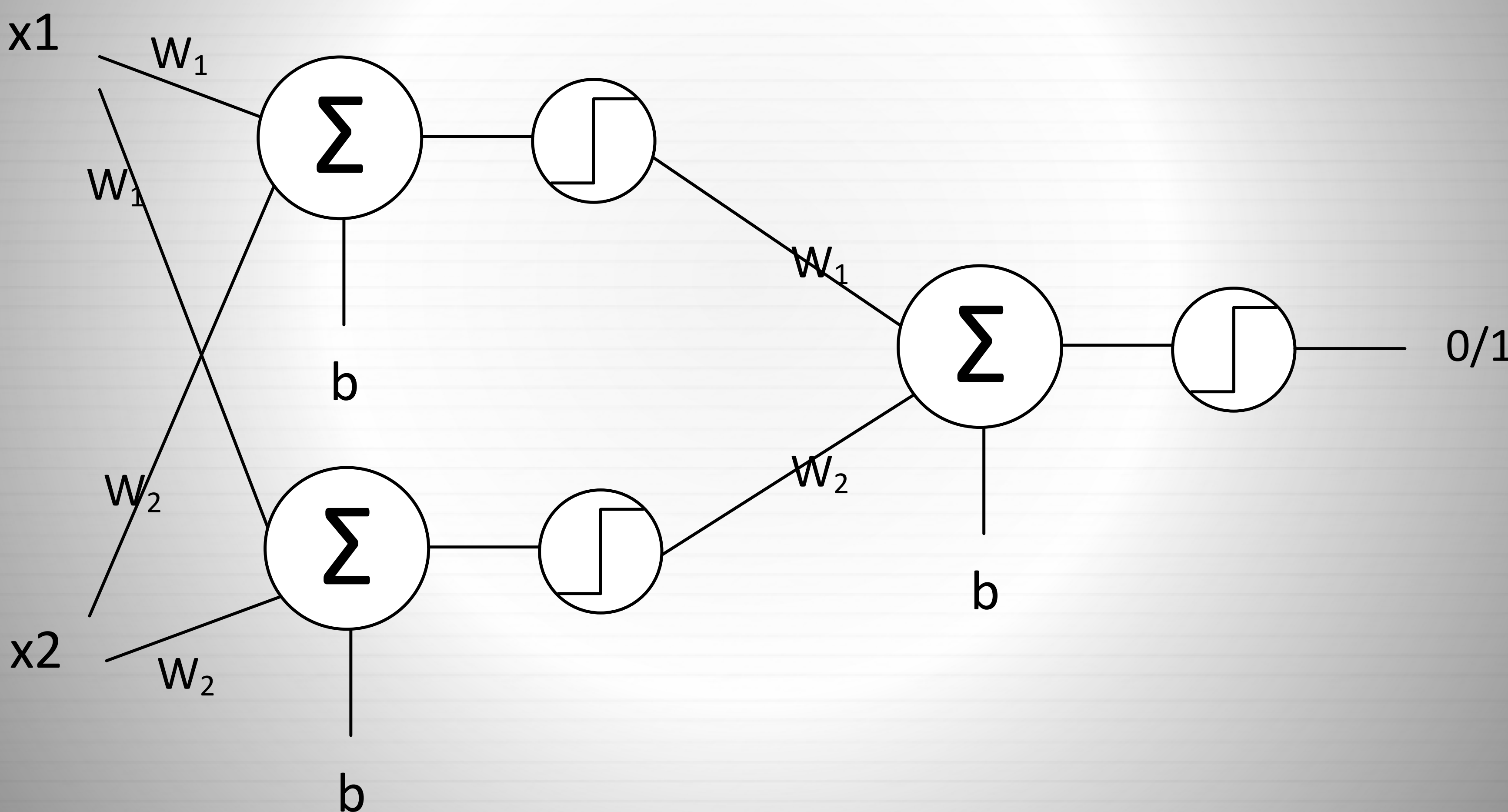


Solution for XOR

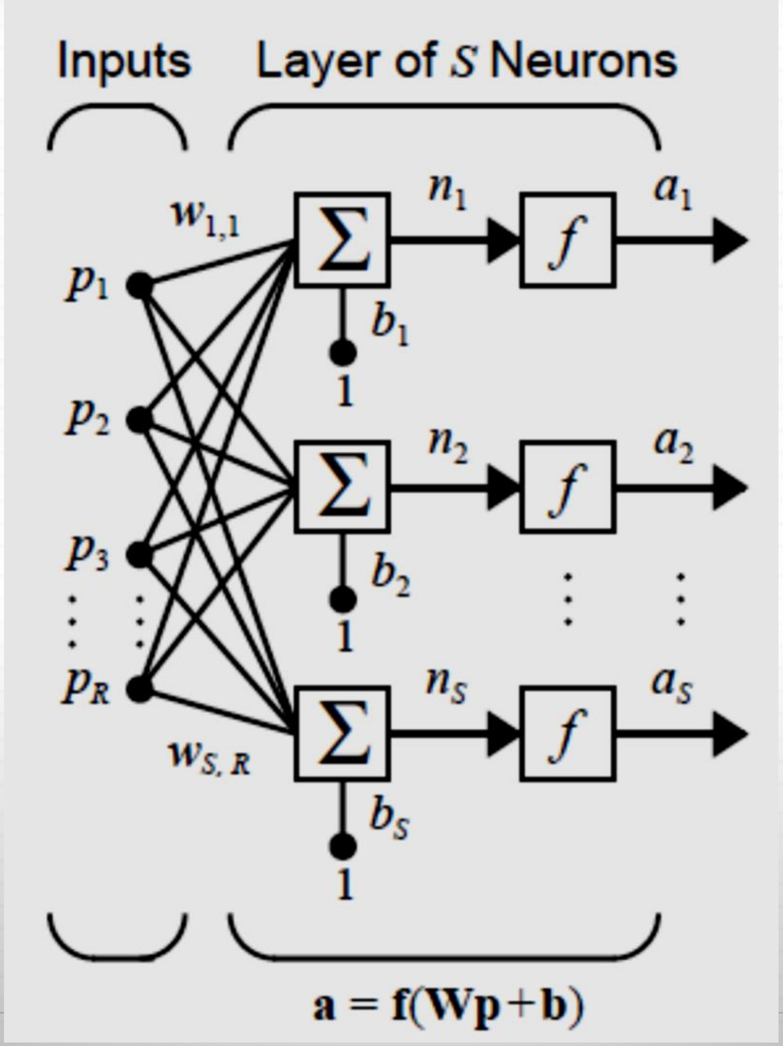
- To classify XOR data we can use two perceptron classifiers.



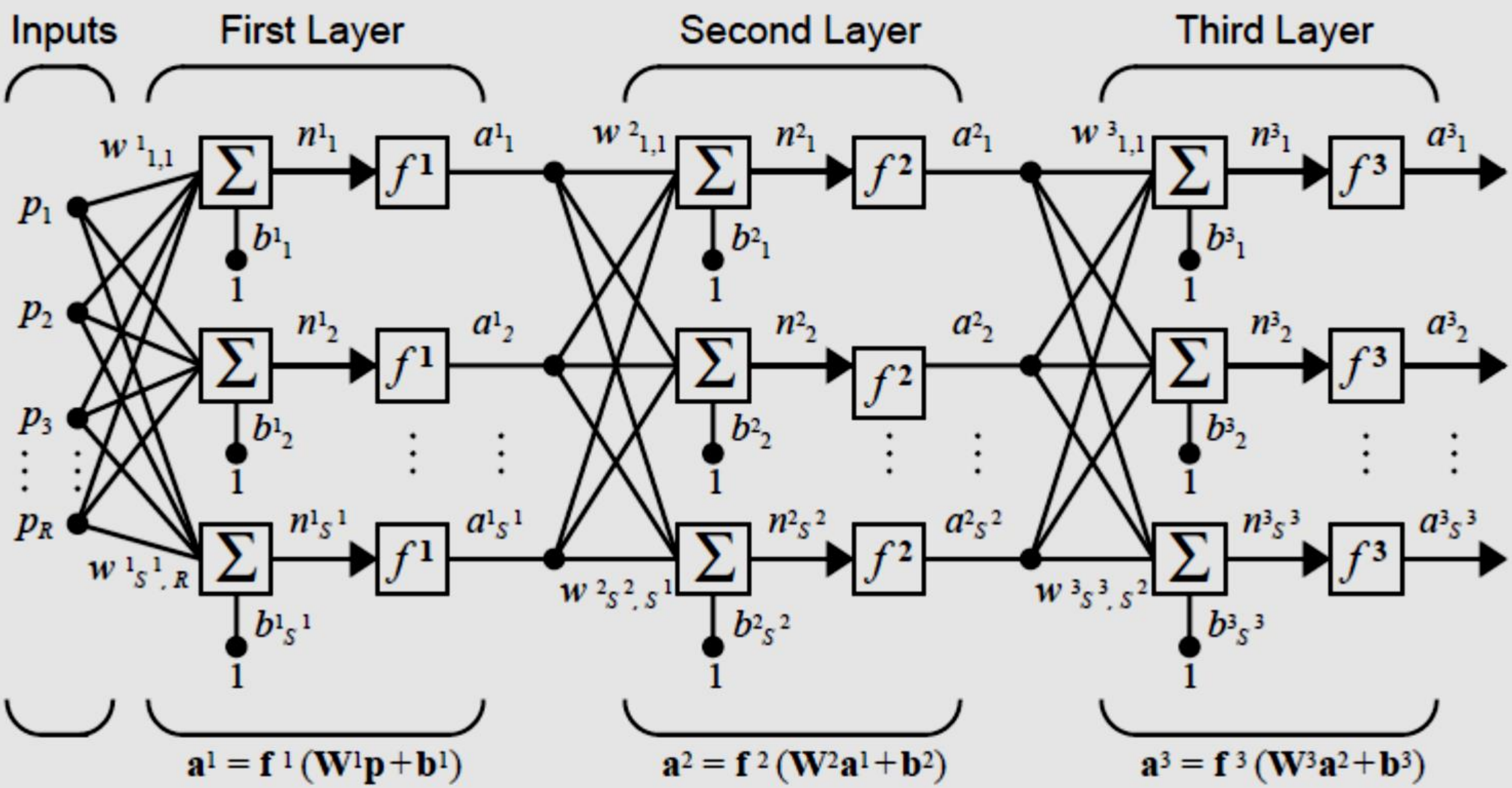
XOR Classifiers



Multi-Perceptron Classifier



Multi-Layer Perceptron



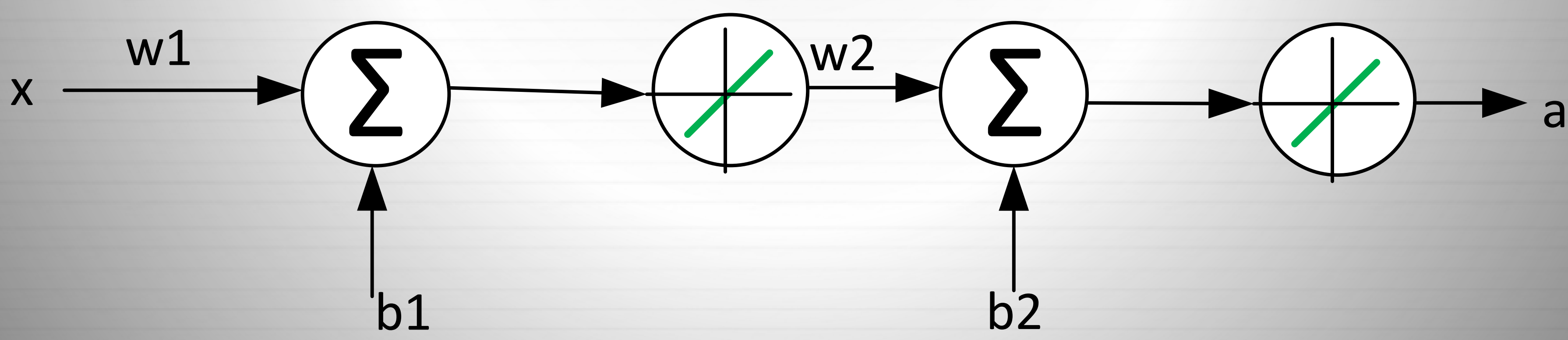
$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

Learning Algorithm

- In training multilayer perceptron, an error term is defined as the square of the difference between the actual output and the true output.
- Next, the impact of each parameter change on minimizing the error is found (using Gradient)
- Finally, the parameters are updated with a scale called ***learning rate.***
- This algorithm is called ***backpropagation***

Learning Example

- Let's consider a very simple neural network with two neurons. The transfer function is ***Linear Transfer***
- The network has only one input.
- The parameters are w_1, b_1, w_2, b_2



Learning Example

- The cost function is: $cost = (t - a)^2$
- The network output is $a = (xw_1 + b_1)w_2 + b_2$
- Partial derivative of cost with respect to w_1 is

$$\frac{\partial cost}{\partial w_1} = 2\left(t - \left((xw_1 + b_1)w_2 + b_2\right)\right) xw_2$$

- This shows the amount of change of w_1 given sample x
- Same calculations should be done for w_2, b_1, and b_2

Learning Rate

- To change the weight parameters, we multiply the Gradient by a scale value called learning rate.
- Learning rate adjusts our step size in approaching the minimum.
- A small step size will make learning slow
- With a big step size, we may skip over a global minimum

When to Update the Weights?

- There are three options is updating the weights:
- *1-* After finding the gradient for each sample
- *2-* At the end of an epoch (using the average of their gradients)
- *3-* After testing a group (batch) of samples (using the average of their gradients)

When to Update the Weights?

- These methods are called:

- **Stochastic Gradient Descent**. ***Update after 1 sample***
- **Batch Gradient Descent**. ***Update after all training samples***
- **Mini-Batch Gradient Descent**. ***1 < Batch Size < Size of Training Set***

Training and Testing Data Sets

- The data set is divided into **Training** and **Test** sets.
- After the network has been trained, we will compute the errors that the trained network makes using ***test set***.
- keep in mind.
  - The test set must never be used to train the neural network. The test set should only be used when the training is complete.
  - Second, the test set must include all situations for which the network will be used.

Optimizations

- Neural Networks do not work well in all cases
- Among the problems that they face is overfitting
- The neural network should be optimized to avoid these problems.

Generalization

- A classifier is modeled as:

$$a^t = g(x^t \mid \theta)$$

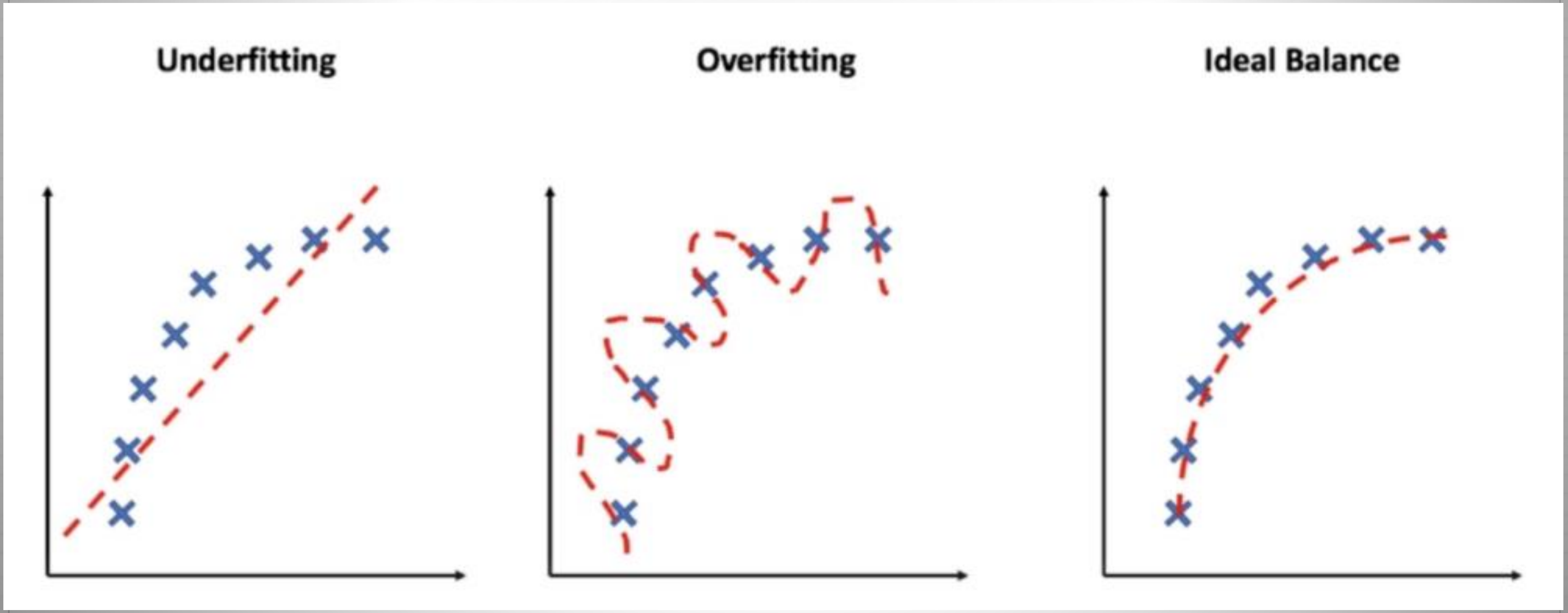
- Function g(.) is estimated using training data
- But the training data is a (generally small) subset of real data, coming from the application.
- Therefore, we want to know if we can generalize the classifier, so that it performs well with any data

Overfitting

- If a model such as g(.) perfectly matches training data points, while performs poorly with other data, we say it overfits the data.

- (Here, we assume the number of data points is limited)

Overfitting



Generalization

- The key strategy for obtaining good generalization is to find the simplest model that explains the data.
- The idea is that the more complexity you have in your model, the greater the possibility for errors.
- In terms of neural networks, the simplest model is the one, that contains the smallest number of parameters (weights and biases), or, the smallest number of neurons.

Generalization

- There are (at least) four different approaches to produce simple networks:
  - growing,
  - pruning,
  - regularization,
  - early stopping.

Growing

- Growing methods start with no neurons in the network and then add neurons until the performance is sufficient.

- Since the network grows gradually, the minimum number of required parameters can be found

Pruning

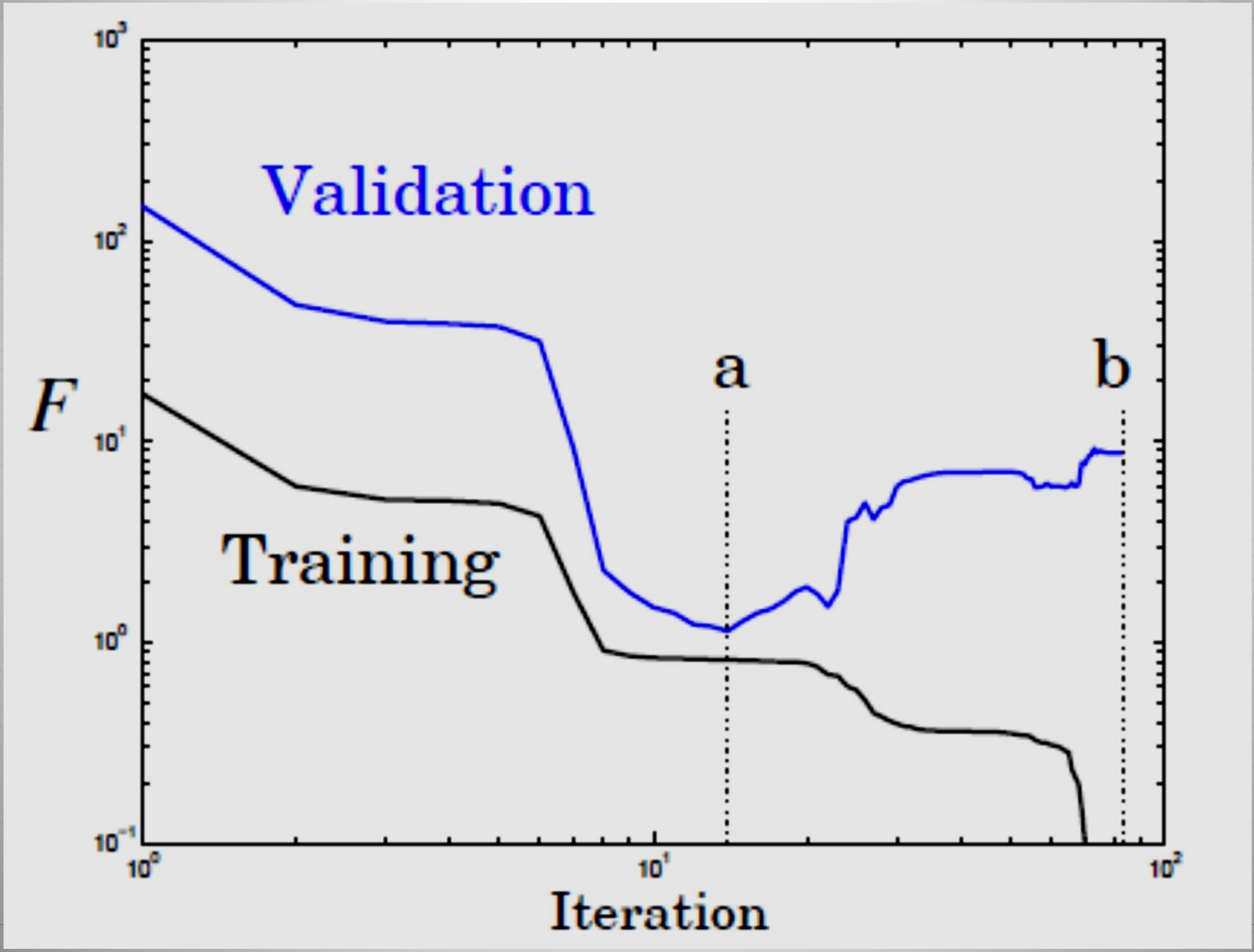
- Pruning methods start with large networks, which likely overfit, and then remove neurons (or weights) one at a time, until the performance degrades significantly.
- Pruning methods are opposite of growing methods.
- The main problem with both growing and pruning methods is, how to grow/prune the network.

Early Stopping

- As training progresses, the network uses more and more of its weights, until all weights are fully used when training reaches a minimum error.
- By increasing the number of iterations of training, we are increasing the complexity of the resulting network.
- If training is stopped before the minimum error is reached, then the network will effectively use fewer parameters and will be less likely to overfit

When to Stop?

- Use a *validation set* to decide when to stop.
  - The available data (after removing the test set) is divided into two parts: a training set and a validation set.
  - The training set is used to determine the weight update at each iteration.
  - The validation set is to measure the error during the training process.
- When the error on the validation set goes up for several iterations, the training is stopped.
- The weights that produced the minimum error on the validation set, are used as the final trained network weights



10^3
10^2
10^1
10^0
10^{-1}
F
Validation
Training
a
b
10^0
10^1
10^2
Iteration

K-Fold Cross Validation

- K-fold cross validation is performed using the following steps:
- Partition the original training data set into k equal subsets. Each subset is called a **fold**. Let the folds be named as $f_1, f_2, \ldots, f_k$.
- For i = 1 to k
  - Keep the fold f_i as Validation set and use all the remaining *k-1* folds as training set.
  - Train the model using the training set and calculate the accuracy of the model using the validation set.
- Estimate the accuracy of your machine learning model by averaging the accuracies derived in all the *k* cases of cross validation.

Regularization

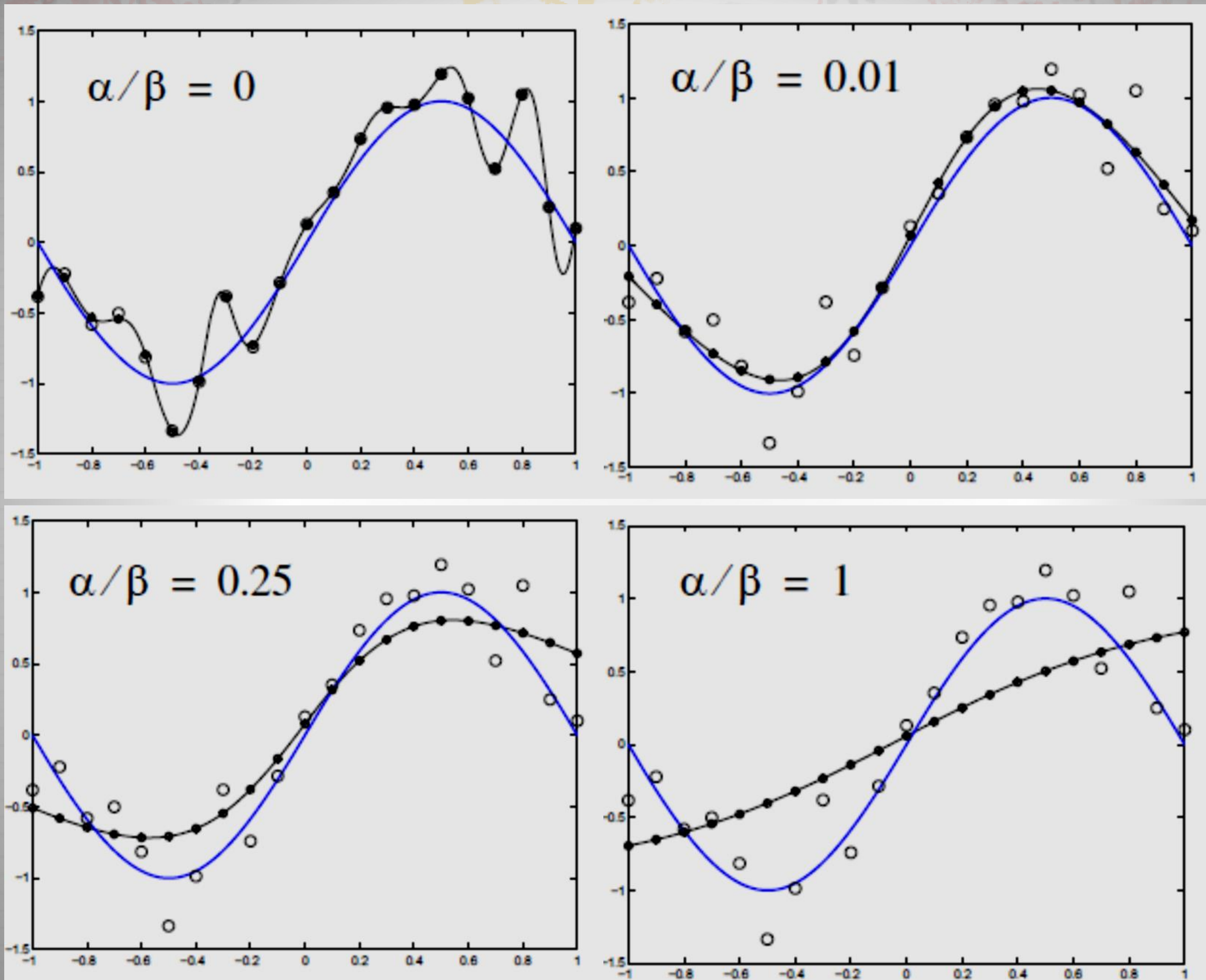
- In ***Regularization*** method, we modify the sum squared error to include a term that reduces network complexity.

- This regularization term can be written as the sum of squares of the network weights (called L2 regularization).

$$\mathrm{F(x)} = \beta E_D + \alpha E_W = \beta \sum_{i=1}^{Q} (t_i - a_i)^2 + \alpha \sum_{i=1}^{n} w_i^2$$

Controlling Complexity

- The ratio α/β controls the effective complexity of the network.
- The larger this ratio is, the smoother the model curve.


α/β = 0
α/β = 0.01
α/β = 0.25
α/β = 1

Summary

- Neural networks are one of the most popular methods in machine learning
- We can use single layer or multi-layer neural networks
- We use training and testing data for learning
- The most important problem in neural networks is overfitting
- When the network is less complex, the probability of overfitting is smaller
- As the solution, early stopping or regularization are mainly used

Research Assignment

- L2 regularization adds sum squared of weights to the cost function.
- We have the option of adding sum of the absolute values of the weights to the cost function.
- This regularization is called L1 regularization
- Discuss the differences between L1 and L2 and how they affect the structure of neural networks

Due date: December 20, 2020

Question?